

**PM: A Distributed Object-Oriented
Operating System**

Franz J. Hauck

May 1993

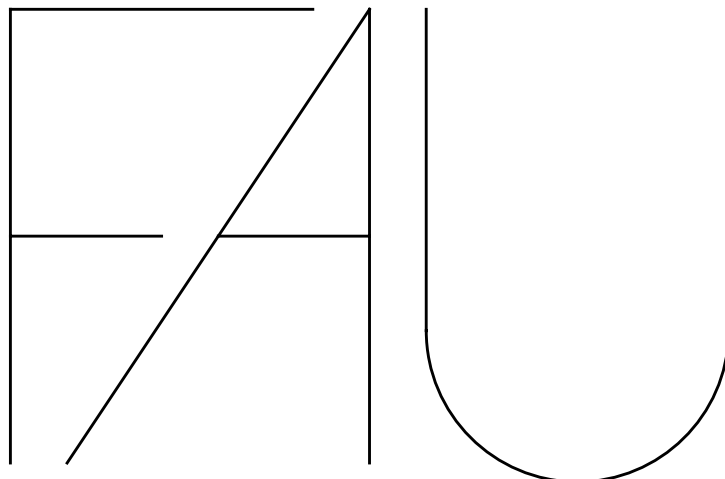
TR-I4-2-93

Technical Report

Computer
Science Department

Operating Systems — IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany



This paper was submitted to the ECOOP'93 Workshop W7
“Workshop on Object-based Distributed Programming”

***PM*: A Distributed Object-Oriented Operating System**

Position Paper — ECOOP'93 Workshop W7

Franz J. Hauck

University of Erlangen-Nürnberg
Department of Computer Science, IMMD 4
P.O. Box 3429, D-91051 Erlangen, Germany
hauck@informatik.uni-erlangen.de
Tel. +49.9131.85.8029

Abstract

The *PM* project is introduced. Its goal is the design of a distributed object-oriented operating system. One of the research activities is the design of an object model for distributed object-oriented programming. This position papers presents the *PM Object Model* approach to distribution, concurrency control and object interaction. The system consists of an application system, a distribution system, and a coordination system which handle orthogonal parts of the object model. The *PM System Architecture* and future work are introduced.

1 Introduction

PM is a project for designing a distributed object-oriented operating system¹. The basic entities to be supported by the operating system are objects. The operating system itself is constructed in an object-oriented way. The major goal of the system design is the adaptability to various classes of general purpose tasks — very special tasks are excepted e.g. numerical computations. *PM* will not be completely implemented, but there will be partial implementations as simulators etc. which will allow a validation of the concepts.

The *PM* project is divided into two major parts: *System Architecture* and *Object Model*. The *System Architecture* deals with the design of components of an operating system which supports a distributed object model. Besides, the components are designed object-oriented using the *PM Object Model*. The *Object Model* deals with the design of a distributed object-oriented language. This language contains typical operating system features as distribution, concurrency, and protection.

This position paper describes the *Object Model* part of the project. The linguistic support for distribution related aspects is discussed and the results as well as needed future work are presented.

2 Object Model

The major goal of *PM* is the design of an adaptable operating system. To support this goal we defined two design criterions:

Uniformity

The object model should be uniform. There should not be two, or even more, different kinds of objects with different semantics as seen in several object-oriented systems: e.g. C++ [EllStr90], *Argus* [Liskov85], *Clouds-Distributed Eiffel* [GunLeB91].

Orthogonality

Orthogonal features of the object model should be described independently. This criterion allows a maximum of reusability because all features can be adapted without affecting others.

1. The *PM* project is supported by the *Deutsche Forschungsgemeinschaft – DFG* – with grant SFB 182.

The object model consist of an application system which defines application objects, of a distribution system which defines the configuration of the application and its distribution, and of a coordination system which defines the coordination of application objects.

2.1 Application System

The basis of the *PM* object model is similar to *Emerald* [Hutchi87]. Objects are the smallest entities for distribution. Nearly everything is an object. There is only one kind of objects, even *Integer* values are first class objects. Objects contain references to other objects. References are bound to named variables which are part of an object. Methods are named components of objects. A method can be invoked when another invocation has access to a reference to the object which contains the method.

Method invocations can be *synchronous* (the caller waits for the result of the invocation) or *asynchronous* (the caller can go on with its computation and can claim the results later). Asynchronous invocations are attached to *handles*. Handles support a blocking and a non-blocking wait operation as well as direct and blocking access to the results of the invocation. Up to now, there is no possibility of using multicasts or different protocols of interactions by application programmers, but the handles are similar to meta-objects and could be an entity to place user defined protocols in the future.

Inheritance is modelled as a special aggregation relation which can be typed and dynamically bound like normal object references [Hauck93]. This allows the distribution of an object of a subclass on the boundary between sub- and superclass.

2.2 Distribution System

In contrast to *Emerald* [Hutchi87] distribution is described in a separate distribution language. Thus, distribution can be changed without affecting the application program. Distribution is inferred by relations between objects. Possible relations are called *collocated* (objects should be placed on the same node), *dislocated* (objects should be placed on different nodes), and *unspecified*. A separate system called *Distribution System* controls these relations. It allows the definition of initial relations between the application objects. The dynamic behavior of distribution is described by *cooperations*. *Cooperations* are part of the distribution system. They are instantiated and deleted by events. Possible events are begin and end of invocation, creation of objects etc. [Fäustle93]. A *cooperation* changes a defined set of relations between certain objects dynamically during its lifetime. Thus, migration of objects can be controlled by *cooperations*.

A second set of relations is used to describe the protection domains of objects. *Collocated* objects are placed in the same virtual address space, *dislocated* objects have to have their own address space. The protection domains are orthogonal to distribution.

2.3 Coordination System

As well as distribution, concurrency control is separated from application programming as much as possible. A *Coordination System* defines the concurrency control inside of the application objects. Coordination objects of the *Coordination System* are attached to application objects and control the internal activities of the attached objects (*cooperative coordination*). It is possible to attach one coordination object to several application objects [Pruy92]. Coordination objects may interact with each other to control a set of objects in defined way.

3 System Architecture

The implementation of the operating system is done by objects modelled as part of the *PM System Architecture*. These objects form a meta-system to the distributed application system supported by the operating system. The meta-system is structured in meta-levels or meta-spaces similar to the *Apertos* approach

(founded with the name *Muse*) [YTMFT91]. The object and memory management of the meta-system is roughly designed in [Kleinöd93], other parts will follow.

4 Future Work

Future work will deal with replication, persistence and transactions. Replication is considered as an important feature for fault tolerance in distributed systems. For the *PM* object model we like to integrate replication as an linguistic feature with full transparency of communication protocols and partial transparency for consistency between the replicas.

Transactions are a possibility to get both persistence and coordination. The coordination by transactions is not suitable for operating system applications. Future work will be on getting objects persistent and recoverable without the restrictive concurrency control of transaction systems.

Runnable implementations are planned as part of a validation of the *Object Model* concepts. These implementations will only model a part of the system and will be simulators of the concepts.

5 References

- [EllStr90] M. A. Ellis, B. Stroustrup: *The annotated C++ reference manual – ANSI base document*; Addison-Wesley, Reading, Mass., USA, 1990
- [Fäustle92] M. Fäustle: *Beschreibung der Verteilung in objektorientierten Systemen*; Dissertation; Arbeitsber. d. IMMD 25(8); University of Erlangen-Nürnberg, IMMD; Sep. 1992
- [Fäustle93] M. Fäustle: “An orthogonal optimization language for uniform object-oriented systems”; *Parallel Comp. Architectures: Theory, Hardware, Software, and Appl.* – SFB Colloquium SFB 182 and SFB 342; A. Bode, H. Wedekind [Eds.], (Munich, Oct. 8-9, 1992); Lecture Notes in Comp. Sci.; Springer, Berlin et.al., to appear 1993 — also available as: Techn. Report TR-I4-6-92; Univ. of Erlangen-Nürnberg, IMMD 4, Oct. 1992²
- [GunLeB91] L. Gunaseelan, R. J. LeBlanc jr.: *Distributed Eiffel – a language for programming multi-granular distributed objects on the Clouds operating system*; Techn. Report GIT-CC-91/50; Georgia Tech., Atlanta, Ga., 1991
- [Hauck93] F. J. Hauck: *Inheritance modelled by aggregation – an approach to typed inheritance relations* – submission to the OOPSLA ‘93 conference; Techn. Report TR-I4-1-93; Univ. of Erlangen-Nürnberg, IMMD 4; Feb. 19, 1993²
- [Hutchi87] N. C. Hutchinson: *Emerald: an object-based language for distributed programming*; Univ. of Washington, Seattle, Wash.; Techn. Report 87-01-01, PhD Thesis; Jan. 1987
- [Kleinöd93] J. Kleinöder: “Object and memory management architecture – a concept for open, object-oriented operating systems”; *Parallel Comp. Architectures: Theory, Hardware, Software, and Appl.* – SFB Colloquium SFB 182 and SFB 342; A. Bode, H. Wedekind [Eds.], (Munich, Oct. 8-9, 1992); Lecture Notes in Comp. Sci.; Springer, Berlin et.al., to appear 1993 — also available as: Techn. Report TR-I4-2-92; Univ. of Erlangen-Nürnberg, IMMD 4, Oct. 1992²
- [Liksov85] B. Liskov: “The Argus Language and System”; *Distributed Systems, Methods and Tools for Specification*; M. Paul, H. J. Siegert [Eds.]; Springer, Berlin et. al., 1985; pp. 343-430
- [Pruy92] R. Pruy: *Cooperative concurrency control*; Techn. Report TR-I4-18-92; Univ. of Erlangen-Nürnberg, IMMD 4, Sep. 1992²
- [YTMFT91] Y. Yokote, F. Teraoka, A. Mitsuzawa, N. Fujinami, M. Tokoro: “The Muse Object Architecture: A New Operating System Structuring Concept”; *SIGOPS* 25(2); ACM, New York, NY, Apr. 1991; pp. 22-46

2. Technical reports are also available on the anonymous ftp server ftp.uni-erlangen.de (131.188.1.43) in the directory: /pub/papers/immd4/InternalReports